AMENDMENTS IN THE TITLE

Please replace the title as follows:

~~SYSTEM AND METHOD FOR PHYSICAL MEMORY ALLOCATION IN ADVANCED OPERATING SYSTEMS~~ **PROGRAMMATICALLY PRE-SELECTING SPECIFIC PHYSICAL MEMORY BLOCKS TO ALLOCATE TO AN EXECUTING APPLICATION**

AMENDMENTS IN THE SPECIFICATION

Please replace the paragraph beginning on page 2, line 4, with the following:

In addition to the major hardware components, a major software (or firmware) component of a data processing system is an Operating System (OS). The OS provides an environment for the execution of programs by providing specific services to the programs including loading the programs into memory and running the program. The OS also provides resource allocation when there are multiple users or jobs running simultaneously. One of the more widely utilized operating systems is Windows® (a trademark of Microsoft Corporation).

Please replace the paragraph beginning on page 3, line 24, with the following:

In a typical data processing system, information is typically loaded in a system memory wherever free space is available. Thus, a virtual address of a block of information usually does not reflect the physical address (or actual address) in the system memory in which the information is actually stored. Thus, present applications that run under the Windows® OS cannot specify/request a specific physical address when polling for allocated memory from the OS. Likewise, the OS does not provide any detail of the allocated memory (i.e., physical address(es) to the applications.

Please replace the paragraph beginning on page 4, line 4, with the following:

Newer operating systems, such as Windows® 2000 are able to address up to 64G (gigabytes) of physical memory, which, as explained above, may be any memory block across the system. The ability to control the exact memory location allocated to an application during testing and/or execution in order to properly predict memory allocation thus becomes extremely important.

Please replace the paragraph beginning on page 4, line 10, with the following:

European patent EP 0 851 353 A1 describes a method of allocating memory space in a main memory of a computer system to a unified memory architecture device and also describes how physical memory can be organized. This is accomplished by getting a linear address range, and mapping physical memory. Also, Microsoft Windows® 2000 provides separate functions to

allocate and/or de-allocate an amount of physical memory, allocate a linear address range, and map the physical memory to the linear address range in an application program. Microsoft Windows® 2000 further provides a driver function to convert a linear address to a physical address. In Microsoft's implementation, when allocating physical memory, the physical memory is locked down and cannot be swapped/exchanged

Please replace the paragraph beginning on page 4, line 21, with the following:

Also, when testing memory allocation in a multi-node configured system such as "NUMA", where memory resources can be located across two different processing systems, for performance reasons, there is a high probability that the Windows® OS (e.g., Whistler and beyond) running on a first system will try to allocate the memory on the same system. However, the OS' memory allocation functions in the multi-node system is not easily predicted because the OS may occasionally allocate the memory of a second system during processing to a process running on a first data processing system.

Please replace the paragraph beginning on page 6, line 20, with the following:

The preferred embodiment of the invention is completed with Windows® 2000 OS and its provided functional features. The invention expands the functionality of the OS to enable different allocation and organization of physical memory besides the regular physical memory allocation functions provided. In one embodiment, specific formulas are utilized to determine the maximum amount of memory to allocate without grabbing too much memory, and memory is allocated in chunks when requested by the application..

Please replace the paragraph beginning on page 7, line 5, with the following:

In the preferred embodiment, the memory allocation routines are a component part of the application housed in memory. In another embodiment, these routines are programmed within the software of the data processing system as a dynamic link library file. In a third embodiment, the memory allocation subroutines, particularly the lock down and allocation/de-allocation processes are completed by a driver that [[are]] is provided the necessary functional software and hardware elements to complete the functions entailed.

Please replace the paragraph beginning on page 11, line 7, with the following:

Illustrated within memory **107** is operating system (OS) **113** and application **115**. Both OS **113** and application **115** are functional, software encoded components. The OS **113** resides in memory **107** on a permanent basis. Application **115** refers to processes being currently run on the data processing system **100** (by the processor **103**) and whose processes temporarily utilize portions of memory. That is, application processes are temporarily loaded into memory **107** when running. OS **113** is responsible for allocation and de-allocation of memory space as needed by application **115**. System calls provide the interface between a process of the application and the OS. According to the preferred embodiment, these system calls include encoded calls for allocation of specific memory blocks to the application process, as described below. The illustrative embodiment provided herein utilizes Windows® 2000 as the OS and takes advantage of the functionality of Windows® 2000, specifically its Advanced Windowing Extensions (AWE) application programming interfaces (APIs) as described below.

Please replace the paragraph beginning on page 17, line 6 , with the following:

Figure 4 illustrates the flow of the general process for performing the specific memory allocation according to the preferred embodiment of the invention. The process begins at block 401 and thereafter proceeds to block 403, which indicates application requesting specific memory allocation and the request being [[past]] passed to the OS. Following, the OS locks down a block of memory equal to the size of memory required by the application as shown at block 405. The memory block is allocated and locked down in a single step utilizing the AWE APIs provided by the OS. The OS then forwards the virtual memory addresses of the memory that has been locked down to the kernel mode driver, which translates the virtual memory addresses into their associated physical addresses and returns the physical addresses to the OS as shown at block 407. In the preferred embodiment, the OS, directed by the MAS, utilizes the standard AWE API calls to accomplish this task. Then, as shown at block 409, MAS completes a check to determine if the physical memory that is locked down fits within the desired range pre-selected by the application developer. The pre-selected range is a parameter that is stored within MAS or the application and provided to the checking process when required. If the check indicates that there is a memory address within the locked down memory blocks that fits

within/on the desired range, then another determination is made at block 411, whether all of the requested specific memory has been locked down. If all the memory has been locked down, then MAS begins a process of de-allocating memory blocks not within the range as shown at block 413. Otherwise, the lock-step process (i.e., memory lock down, address translation, MAS checking for addresses within range) is repeated until physical memory runs out (or the maximum memory to allocate is reached) as indicated at block 415.

Please replace the paragraph beginning on page 19, line 25, with the following:

Alternatively, the method of sorting though the allocated memory blocks is completed with a Window-like processing. When Windows® OS allocates AWE physical pages, there is a certain pattern to the arrangement of the physical pages. Since the basic algorithm does not provide this pattern, an additional routine is coded within MAS.dll to organize the physical pages in a manner, which keeps in the spirit of the Windows®-like pattern. This organization/arrangement helps to make physical address arrangement appear to have been allocated (and thus arranged) by Windows® OS, while still maintaining the desired range.

Please replace the paragraph beginning on page 20, line 6, with the following:

Further, the sorting and Windows®-like routines may also be adjusted in various ways for time concerns. That is, the Windows®-like routine could limit its search for a good Windows®-like match, and the sorting routine could be set to "mostly" sort the pages instead of completely sorting the pages.

Please replace the paragraph beginning on page 22, line 9, with the following:

The invention is mainly concerned with allocating physical memory for software. However, the MAS.dll may be included within other tools implemented in software applications and may be extended to various other subsystems besides memory in the Windows® 2000_environment. One major advantage of using this invention is that a user can lock down specific physical memory locations in a particular range of physical addresses for software. Additionally, the invention finds particular applicability to testing procedures for memory and memory operation, since the tester (designer of the test application) is able to isolate specific areas/blocks/addresses of memory for testing.

Please replace the paragraph beginning on page 17, line 6, with the following:

Figure 4 illustrates the flow of the general process for performing the specific memory allocation according to the preferred embodiment of the invention. The process begins at block 401 and thereafter proceeds to block 403, which indicates application requesting specific memory allocation and the request being [[past]] passed to the OS. Following, the OS locks down a block of memory equal to the size of memory required by the application as shown at block 405. The memory block is allocated and locked down in a single step utilizing the AWE APIs provided by the OS. The OS then forwards the virtual memory addresses of the memory that has been locked down to the kernel mode driver, which translates the virtual memory addresses into their associated physical addresses and returns the physical addresses to the OS as shown at block 407. In the preferred embodiment, the OS, directed by the MAS, utilizes the standard AWE API calls to accomplish this task. Then, as shown at block 409, MAS completes a check to determine if the physical memory that is locked down fits within the desired range pre-selected by the application developer. The pre-selected range is a parameter that is stored within MAS or the application and provided to the checking process when required. If the check indicates that there is a memory address within the locked down memory blocks that fits within/on the desired range, then another determination is made at block 411, whether all of the requested specific memory has been locked down. If all the memory has been locked down, then MAS begins a process of de-allocating memory blocks not within the range as shown at block 413. Otherwise, the lock-step process (i.e., memory lock down, address translation, MAS checking for addresses within range) is repeated until physical memory runs out (or the maximum memory to allocate is reached) as indicated at block 415.